A Gossip-based Service for Failure Detection and Resource Management In Heterogeneous, Distributed Systems

A. George and R. Subramaniyan

High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL

{george,subraman}@hcs.ufl.edu
www.hcs.ufl.edu

Contents

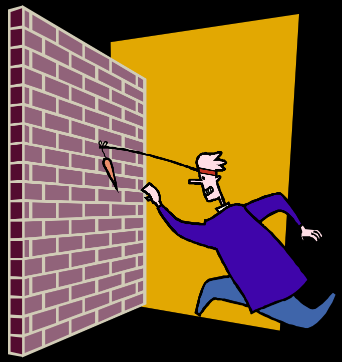
Introduction

- Clusters and cluster-based computational grids are the HPC system of choice for many applications
- Clusters built as a network of workstations from COTS-based components possess an attractive performance vs. cost ratio
- However, as systems scale, so do odds and sources of failures, as does difficulty in monitoring and managing resources

- Systems also becoming increasingly heterogeneous
- **Primary focus of this talk:** Overview of research at UF on efficient techniques for failure detection and resource management in scalable, heterogeneous, distributed systems

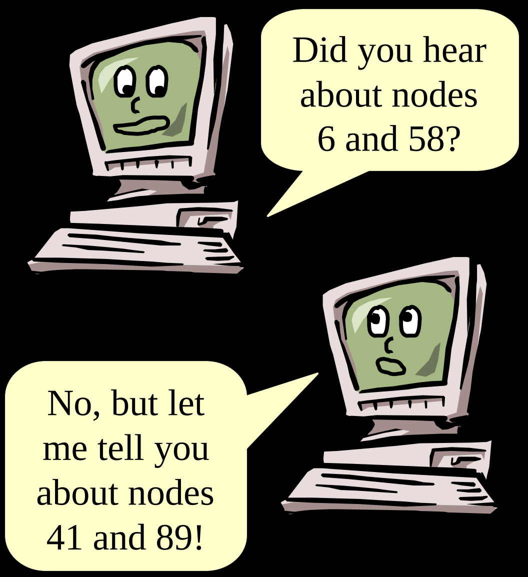
Motivations

- Distributed and heterogeneous nature of clusters makes scalable failure detection and performance a key challenge
- Potential of gossip methods for resource monitoring, failure detection, consensus, etc., scaling with system size
- Gossiping does not critically depend upon any particular network node, path, link, or message
- Past research demonstrating high-speed, low-overhead dissemination and sharing of system state information
- No single point of failure; more efficient than group communication techniques; can be very responsive



Gossip Concepts

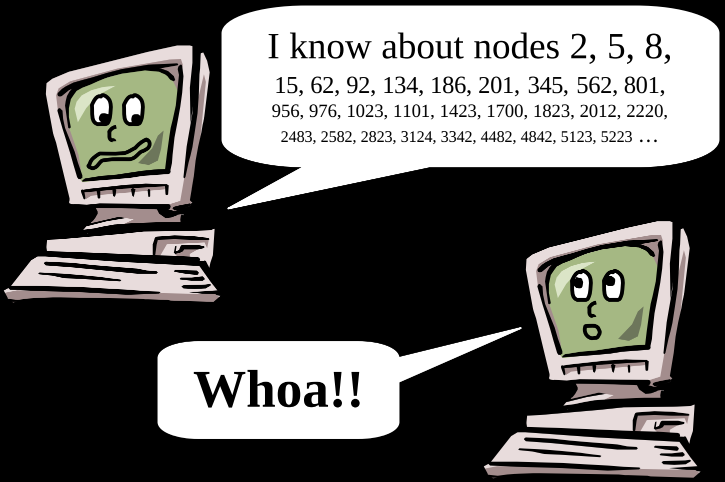
- Early on, used primarily for consistency management of replicated databases, reliable multicast and broadcast
- Nodes communicate according to some underlying randomized or deterministic algorithm
- Every node shares its information with any other node in the system periodically
- Information shared depends upon the service (can be liveness, network load, CPU load, memory load, etc.)

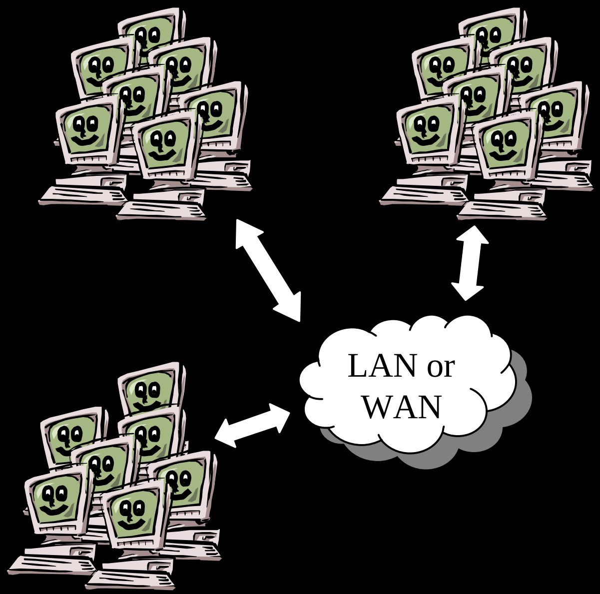


Layered Gossip

Even gossiping is not scalable at first

- Large number of nodes means there is much information to share in each gossip
- Large networks take many rounds of gossiping to completely spread the information



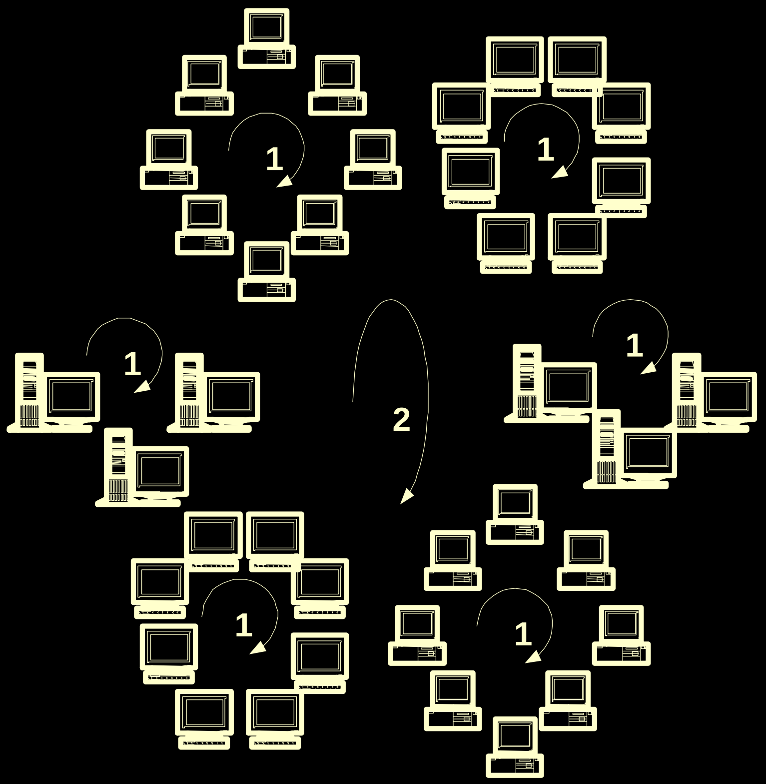


Layered schemes provide scalability

- Network is divided into groups
- Nodes gossip frequently with other nodes in the same group
- Gossip messages are passed between groups less frequently

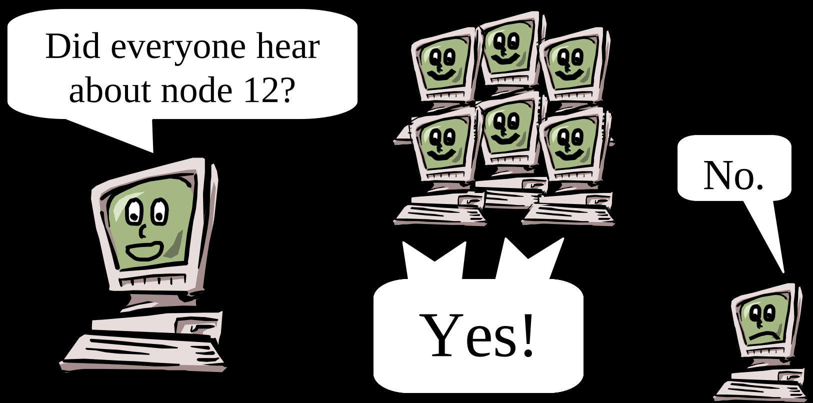
Layered Gossip (cont.)

- Divide and conquer approach
- Nodes in the system are divided into groups
- Groups are arranged in a hierarchical fashion to form the leaves of a ‘**Gossip Tree**’
- Consensus is reached in the lowest group (L1) and propagated to the rest
  - ✓ *For a two-layer system, ‘L1 Gossip’ is intra-group gossip*
  - ✓ *‘L2 Gossip’ is inter-group gossip*



Example of 2-layer system with L1 and L2 gossip

Consensus

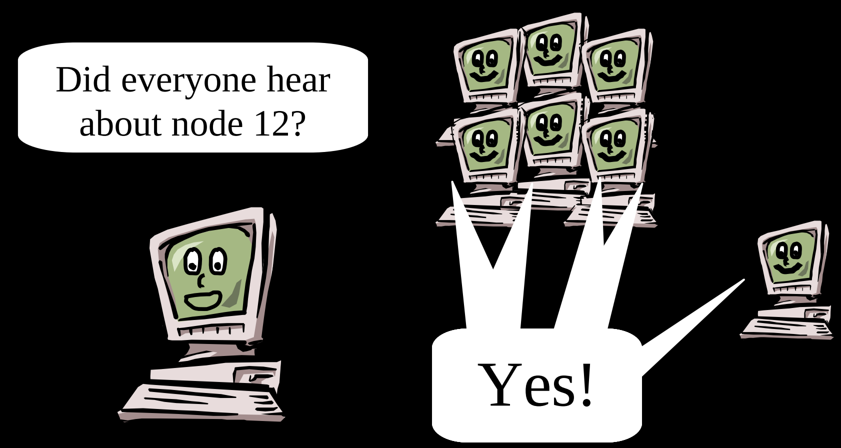


But, gossiping can result in an uneven spread of information

- ➢ Some nodes may detect a failure before others
- ➢ False failure detections may result if information spreads too slowly

Solution: **Consensus** *(critical for failure detection; not necessarily for resource monitoring)*

- ➢ Consensus is reached when a majority of nodes detects the same failure
- ➢ Consensus information is added to the gossip messages
- ➢ Must be performed in distributed fashion to be scalable and avoid SPOF

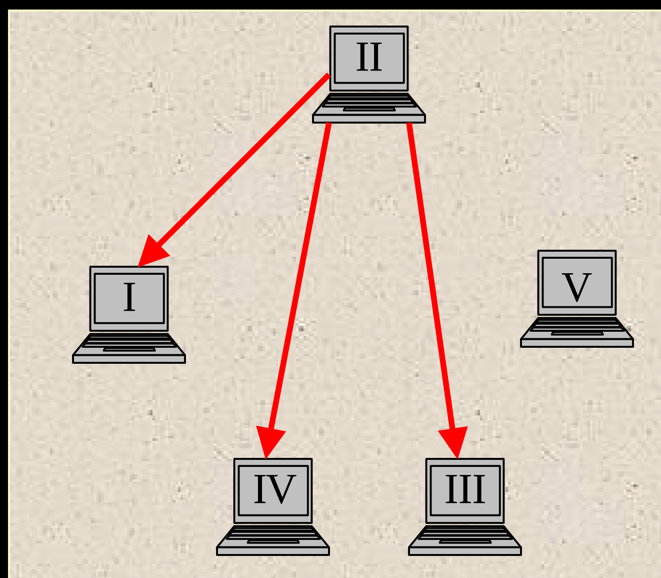


Layered communication also used to support scalable consensus

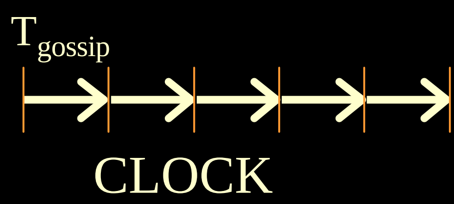
Background

Failure Detection Service

- T_{gossip} or gossip time - interval between two gossip messages
- $T_{cleanup}$ or cleanup time - interval after which a node's failure is suspected
- $T_{consensus}$ or consensus time - interval after which consensus is reached about a failed node



e.g. 5-node system with $T_{cleanup} = 5 \times T_{gossip}$



No. of cycles

5

Node V status

Message from II? NO

Is time since last message = $T_{cleanup}$? YES

Suspect II? YES

Background

Data Structures

- **Gossip list** - vector containing number of T_{gossip} intervals since last heartbeat, for each node
- **Suspect vector** whose i^{th} element is set to '1' if node i is suspected, otherwise it is set to '0'
- The suspect vectors of all the n nodes together form a **suspect matrix** of size $n \times n$
- **Livelist** - vector maintaining the liveness information of all the nodes in the system
- Local suspect matrix and gossip list updated based on received suspect matrix and gossip list

- Gossip list and suspect matrix exchanged every T_{gossip}

Background

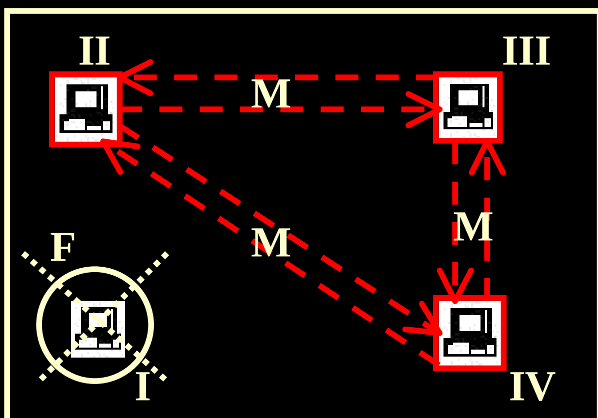
Consensus and Failure Detection

- Consensus reached on the state of node *j* if each element in column *j* of suspect matrix contains a ‘1’

Sample 4-node system

M - Gossip message (*Gossip list + Suspect Matrix*)

F - Failed node



Data structures maintained in one live node

$T_{cleanup}$ = 100ms

I	II	III	IV
110	75	30	0

Gossip list

I	II	III	IV
1	0	0	0

Suspect Vector

Maintained locally not transmitted

	I	II	III	IV
I	0	0	0	0
II	1	0	0	0
III	1	0	0	0
IV	1	0	0	0

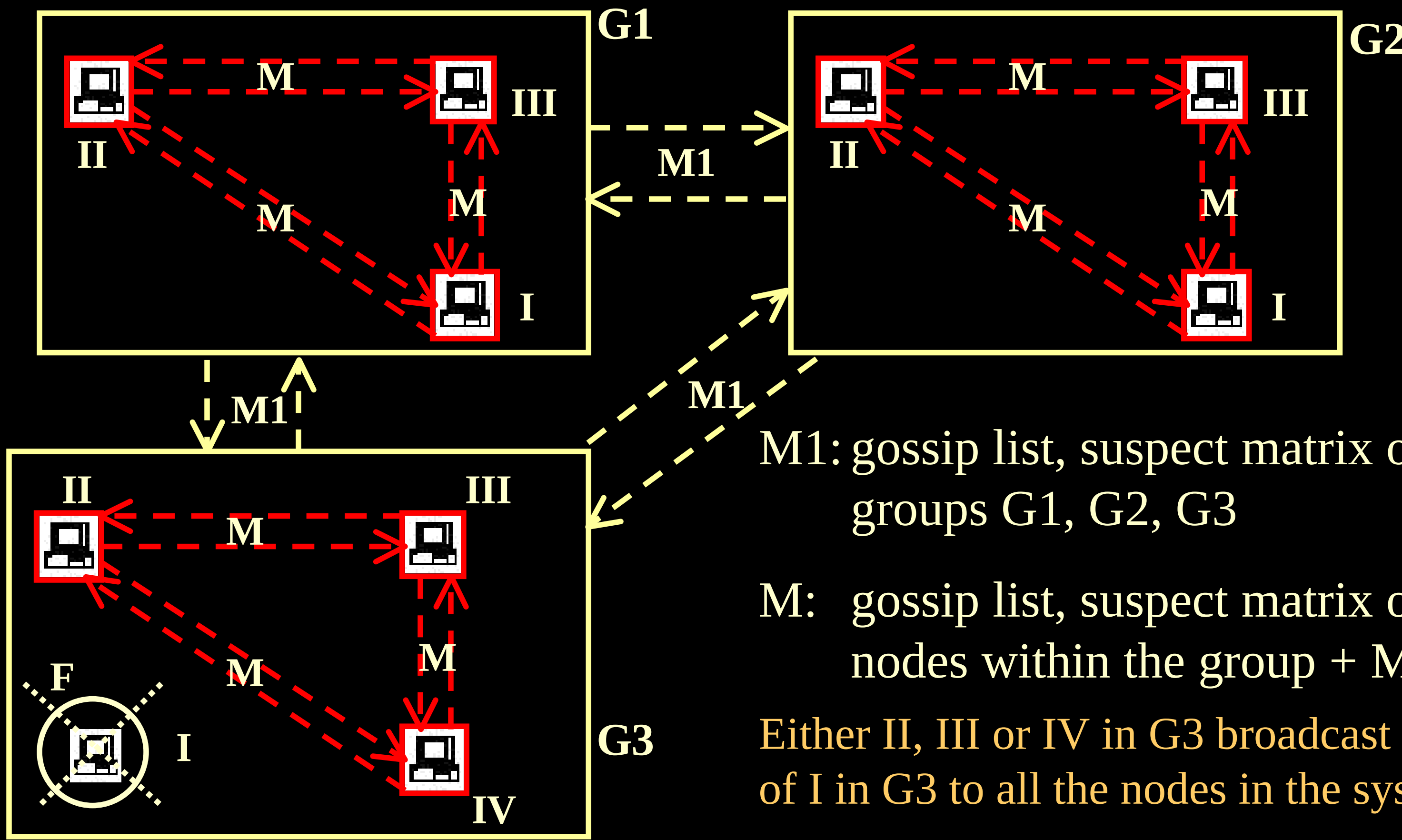
Suspect Matrix

Background

Failure Detection with Layered Gossiping

- Nodes within a group take turns to communicate group information to the other groups

e.g. 2-layered system with 10 nodes divided into 3 groups

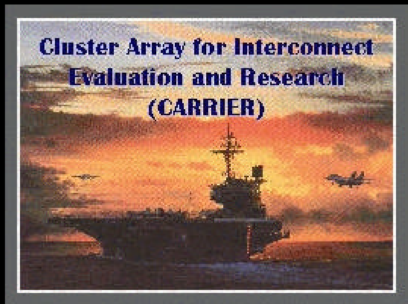


M1: gossip list, suspect matrix of the groups G1, G2, G3

M: gossip list, suspect matrix of the nodes within the group + M1

Either II, III or IV in G3 broadcast the failure of I in G3 to all the nodes in the system

Experimental Testbed



CARRIER: A High-Performance Computer for Architecture, Network, and System Research

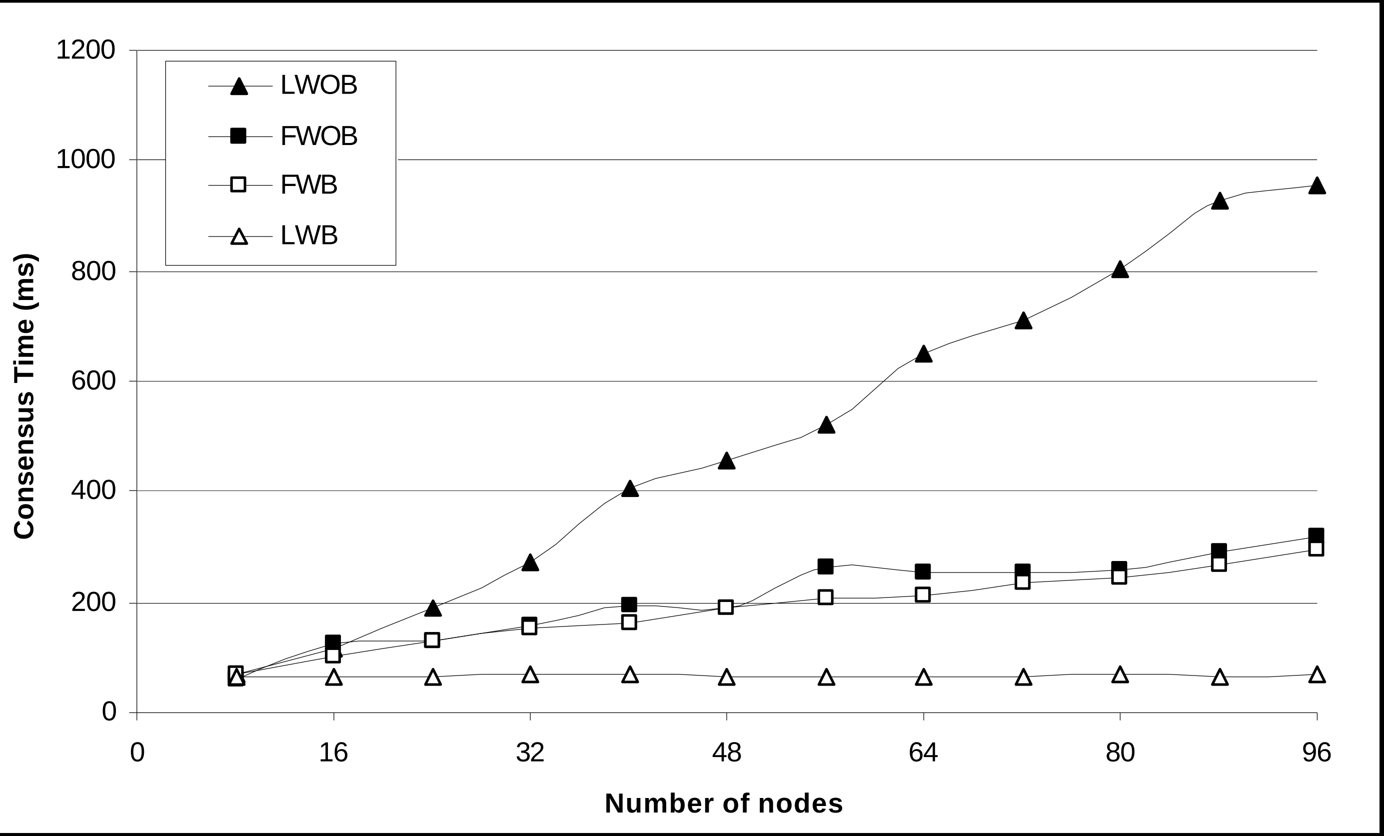
Computational Grid Supercomputer

- 376 Pentium-compatible CPUs
- 240 networked nodes
- 50 GB main memory
- 3.1 TB storage
- Data networks up to 5.3 Gb/s
- PCI64/66 support (i.e. 4×PCI)

For these experiments:

- Up to 96 nodes employed
- Gossip messages sent over control network (switched Fast Ethernet)

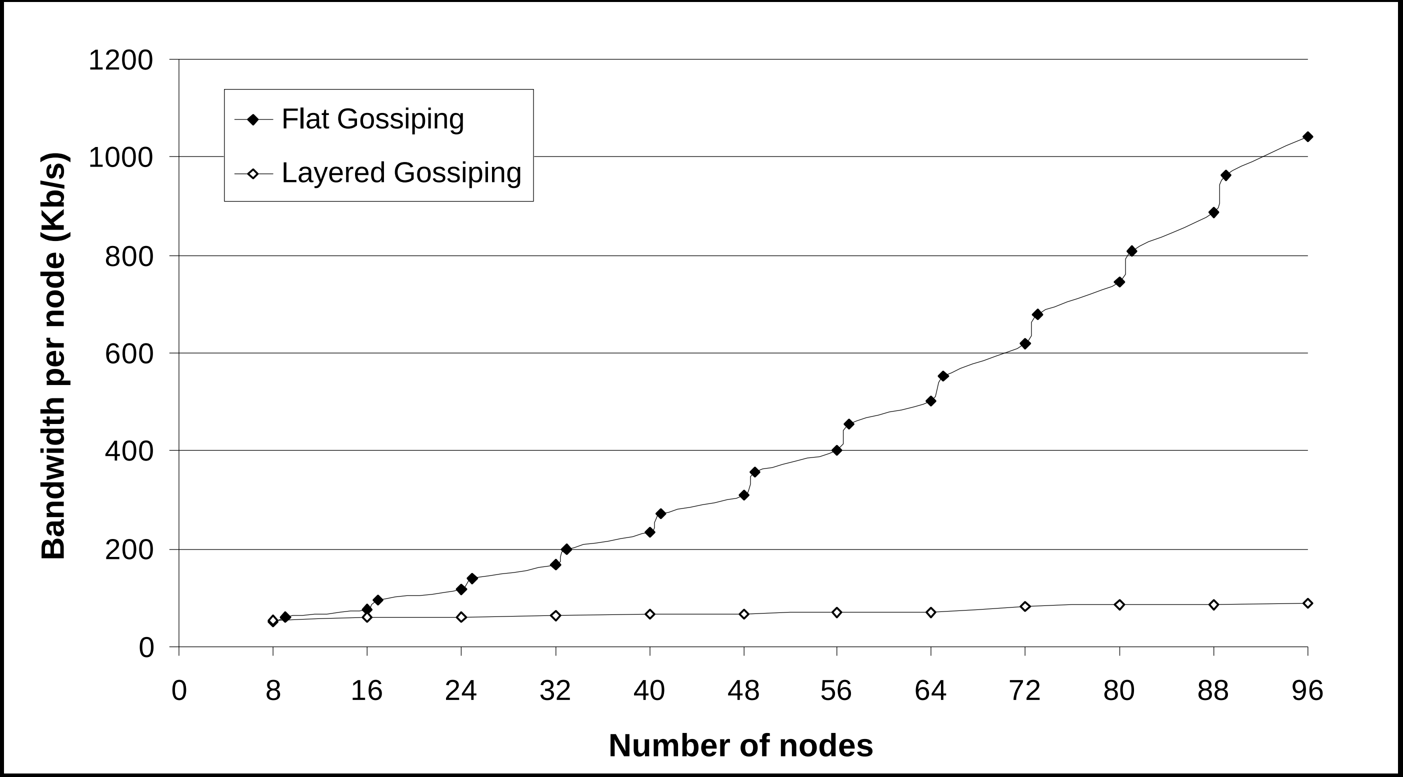
Consensus Time



- T_{gossip} = 10ms
- Flat gossiping uses round-robin (RR) here
- For layered architecture, L1 is RR and L2 is random
- Group size set to 8

- LWB scales almost ideally
- LWOB is least scalable in this configuration
  - attributed to the limited scalability of L2 gossip with fixed group size
  - however, small number of large groups will reach consensus much faster in LWOB than will (as here) a large number of small groups

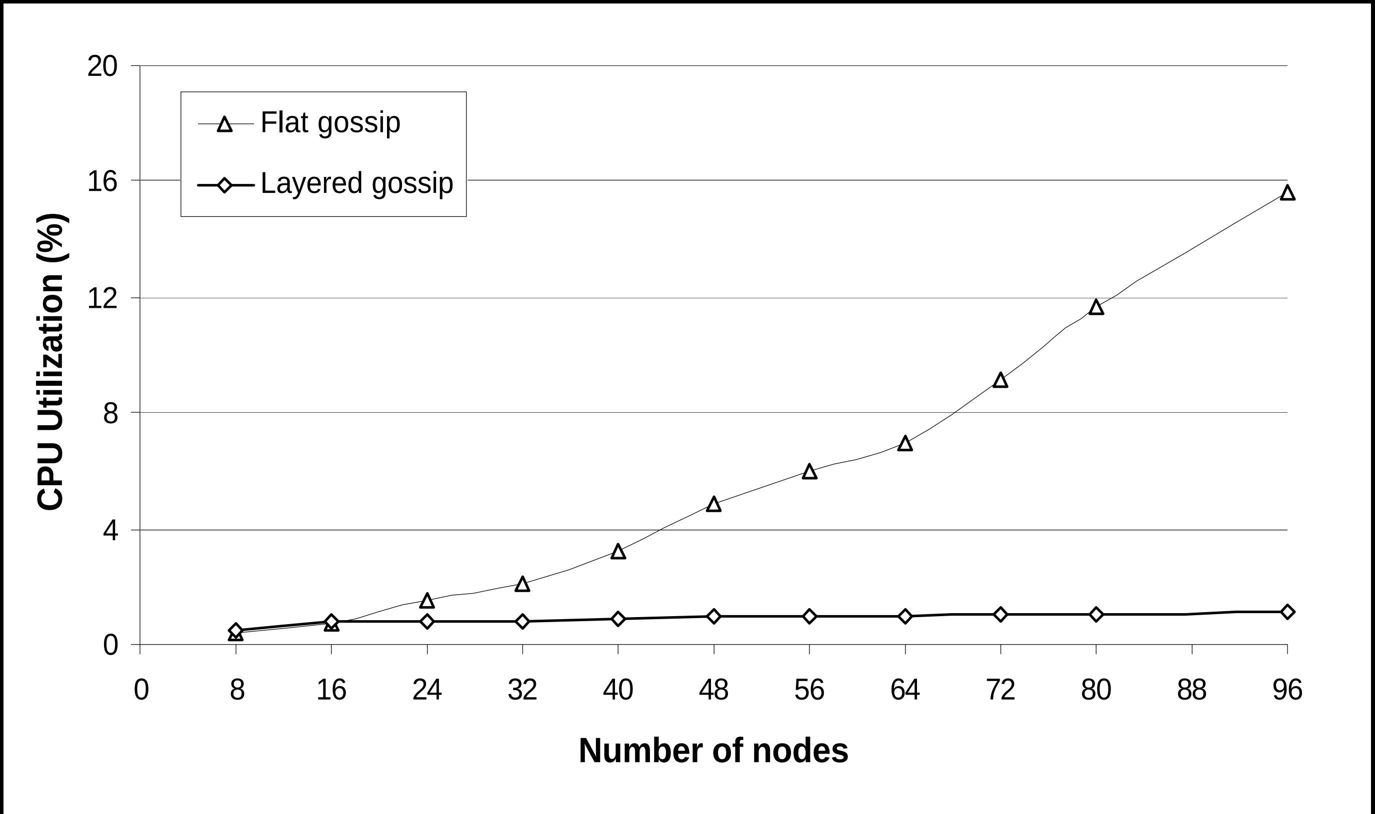
Network Utilization



- T_{gossip} = 10ms
- Flat gossiping uses RR here
- For layered architecture, L1 is RR and L2 is random

- ➢ Layered structure achieves significantly better scalability with modest magnitude by distributed communication with smaller data structures for transmission
- ➢ e.g. with 96-node system, layered service requires only about 10% of network bandwidth utilization associated with flat service

CPU Utilization



- T_{gossip} = 10ms
- Flat gossiping uses random
- For layered architecture, L1 and L2 are both random

- Again, layered structure achieves significantly better scalability with modest magnitude via processing of smaller data structures
- e.g. with 96-node system, layered service requires only ~7% of the CPU utilization required by flat service

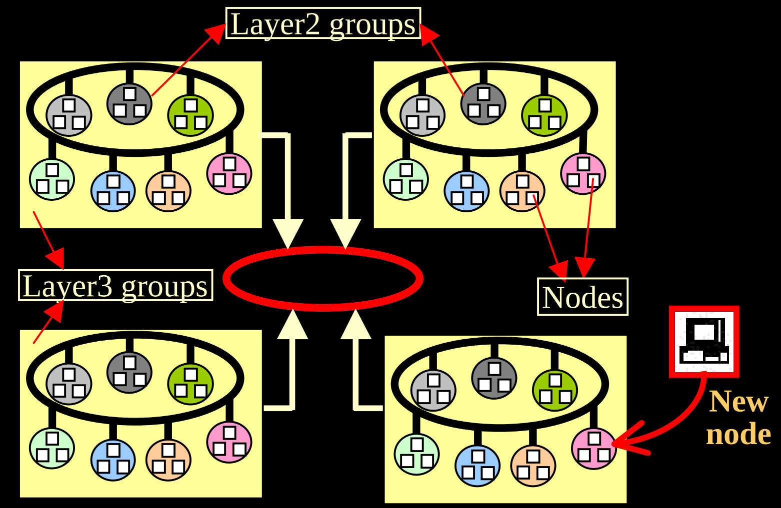
Present Research

- Extensions to failure detection service
  - ✓ Make service more complete, correct and scalable
  - ✓ Aids for dynamic system reconfiguration
- Failure timing and comparison of failure detection services
  - ✓ Determine dependencies of applications on failure detection services and model execution time analytically
  - ✓ Survey of various failure detection services to find best fit based on characteristics and execution time of application
- Gossip-based resource monitoring service
  - ✓ Ascertain state of system resources
  - ✓ Support for load balancing and scheduling services

Failure Detection Service

Protocol Extensions

- Previously, our layered failure detection service supported only two layers, limiting scalability
- Support for *any number of layers* now provided for large-scale systems



e.g. 3-layered system with 84 nodes

- Issues like group failures, network partitions challenging the correctness of the service are addressed and solved
- A new node-insertion mechanism added to improve the dynamic scalability of the system

Resource Utilization

Analytical Formula

- Scalability of enhanced gossip service is verified experimentally
- Formulae developed to project bandwidth per node for very large systems
- Formulae are based on size of Ethernet header, gossip packets, and system configuration

$$B_{layered} = \sum_{i=1}^{l} L_i \times f_i$$

$$L_j = 46 - (l+1) + j + \sum_{k=j}^{l} (g_k + 1)\left(\left\lceil \frac{g_k}{8} \right\rceil + 1\right)$$

g_k : group size in layer k

L_j : length of j^{th}-layer gossip packet

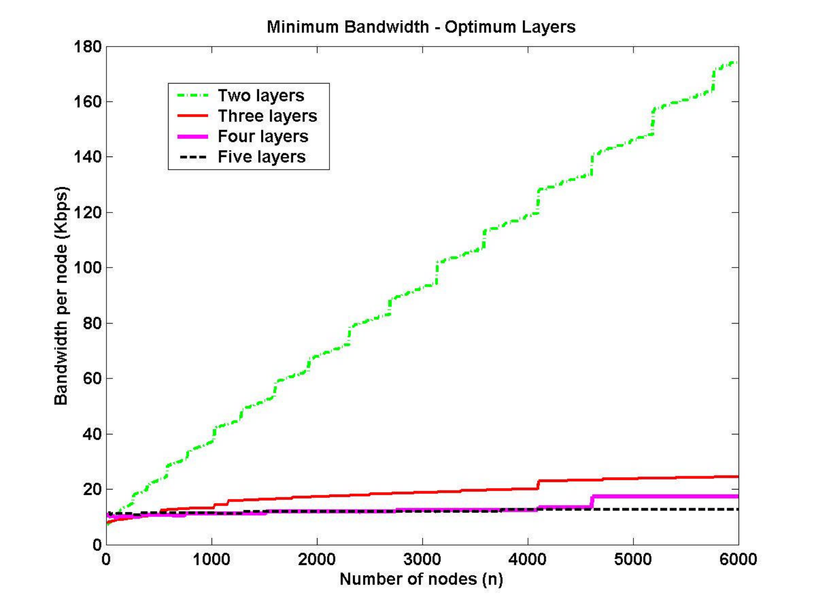
f_i : frequency of i^{th} layer gossip message

B : bandwidth utilization per node

l : number of layers

Resource Utilization

Optimum System Configuration



Generalized formula to calculate group sizes giving approximate minimum bandwidth

- 'L' layered system with 'n' nodes
  - *g_1 = next higher multiple of 8 of ('L' th root of system size)*
  - *g (for other layers) = ' L -1' th root of 'n ÷ g_1'*

Example: A system with 812 nodes and 4 layers

g_1 = next higher multiple of 8 of [812 ^ (1/4)] = 8

$g_2 = g_3$ = (812 ÷ 8)^(1/3) = 5

- The number of layers used to get minimum bandwidth overhead is based on system size
  - * **8 > n < 64 (8 × 8) :** 2 layers
  - * **64 >= n < 512 (8 × 8 × 8) :** 3 layers
  - * **512 >= n < 4096 (8 × 8 × 8 × 8) :** 4 layers
  - * **4096 >= n < 32768 (8 × 8 × 8 × 8 × 8) :** 5 layers
- e.g. Minimum bandwidth per node in a 6000-node system
  - **175 Kbps for 2-layered system ; 11 Kbps for 5-layered system**
- Similarly, for CPU utilization at minimum, requires system configuration to follow trend above as it closely follows pattern of network utilization

Failure Detection Services

Comparisons

- Goal: execute applications as fast as possible despite failures
  - ✓ Determine effects of failures on application
  - ✓ Determine qualities of failure detectors related to these effects
  - ✓ Examine failure detection services to show effects and tradeoffs
- Failure detection services being compared
  - ✓ Gossip – Stand-alone, high-speed, low-level failure detector
  - ✓ Condor – Specialized high-throughput scheduling environment
  - ✓ CORBA – Fault-tolerant object management middleware
  - ✓ Globus – Grid computing middleware
  - ✓ PVM – Cluster computing middleware
- Failure detection services can be categorized in terms of
  - ✓ Method of getting host information – ‘Push’ or ‘Pull’
  - ✓ Failure detection scheme – ‘Centralized’ or ‘Distributed’
  - ✓ Passive versus active
  - ✓ Consensus

Failure Detection Services

Implications of Failures

- Factors that affect application performance
  - ✓ Failure detection time
  - ✓ Checkpointing intervals
  - ✓ Reconfiguration time
  - ✓ Workload redistribution

c = time between checkpoint and failure

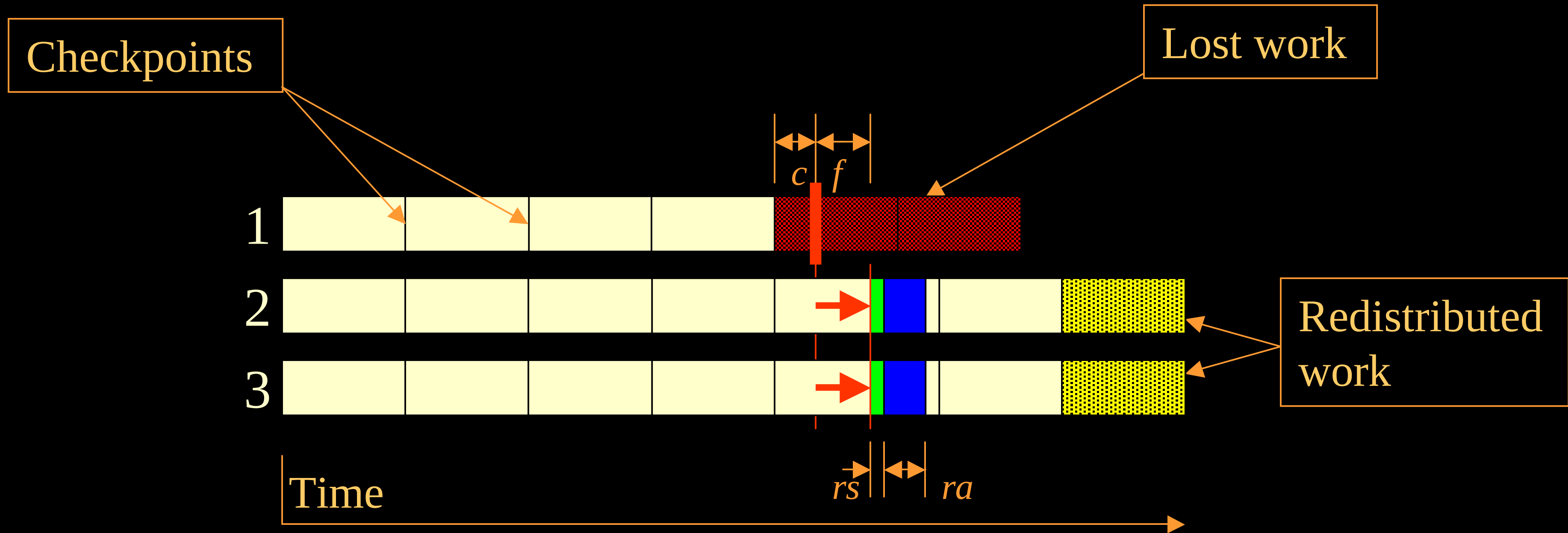
f = failure detection time

rs = system reconfiguration time

ra = application reconfiguration time

= host failure

= failure detection in progress



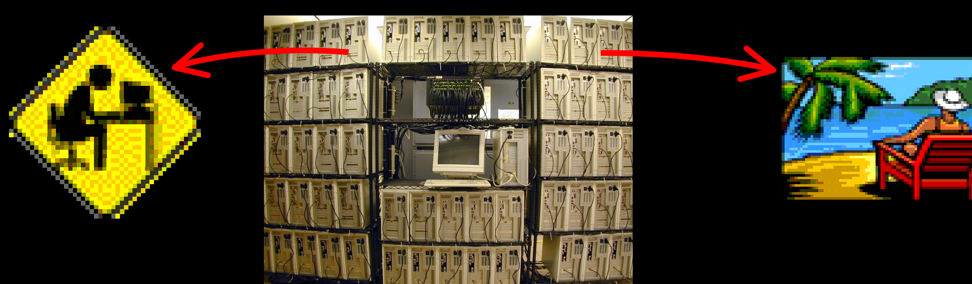
Service on a Service

Gossip-based Resource Monitoring

- Gossip failure detection service is efficient, resilient, and scalable
- But still incurs overhead, and such a service does not reduce computational intensity of the application using it
- Why not piggyback some other system information along with liveness information for efficient dissemination?
- Key Idea: build gossip-based resource monitoring service on top of failure-detection service
- Dependable and scalable approach
- More utility for less price!

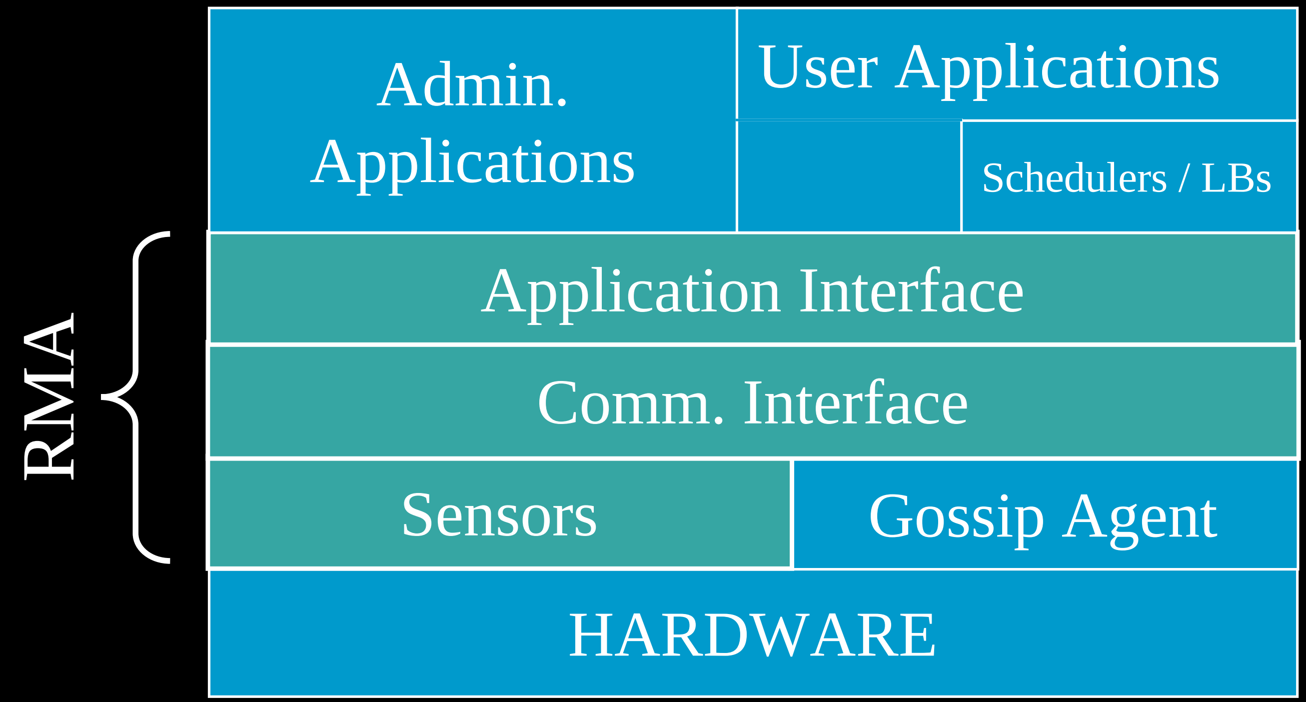
Resource Monitoring

- Useful for detecting and disseminating state of available resources, overloaded conditions
- Critical low-level service for load balancing and scheduling by middleware services and applications
- Essential for system administrators to achieve a single system image of nodes administered
- Source of information regarding resource usage and performance of nodes



Resource Monitoring Service

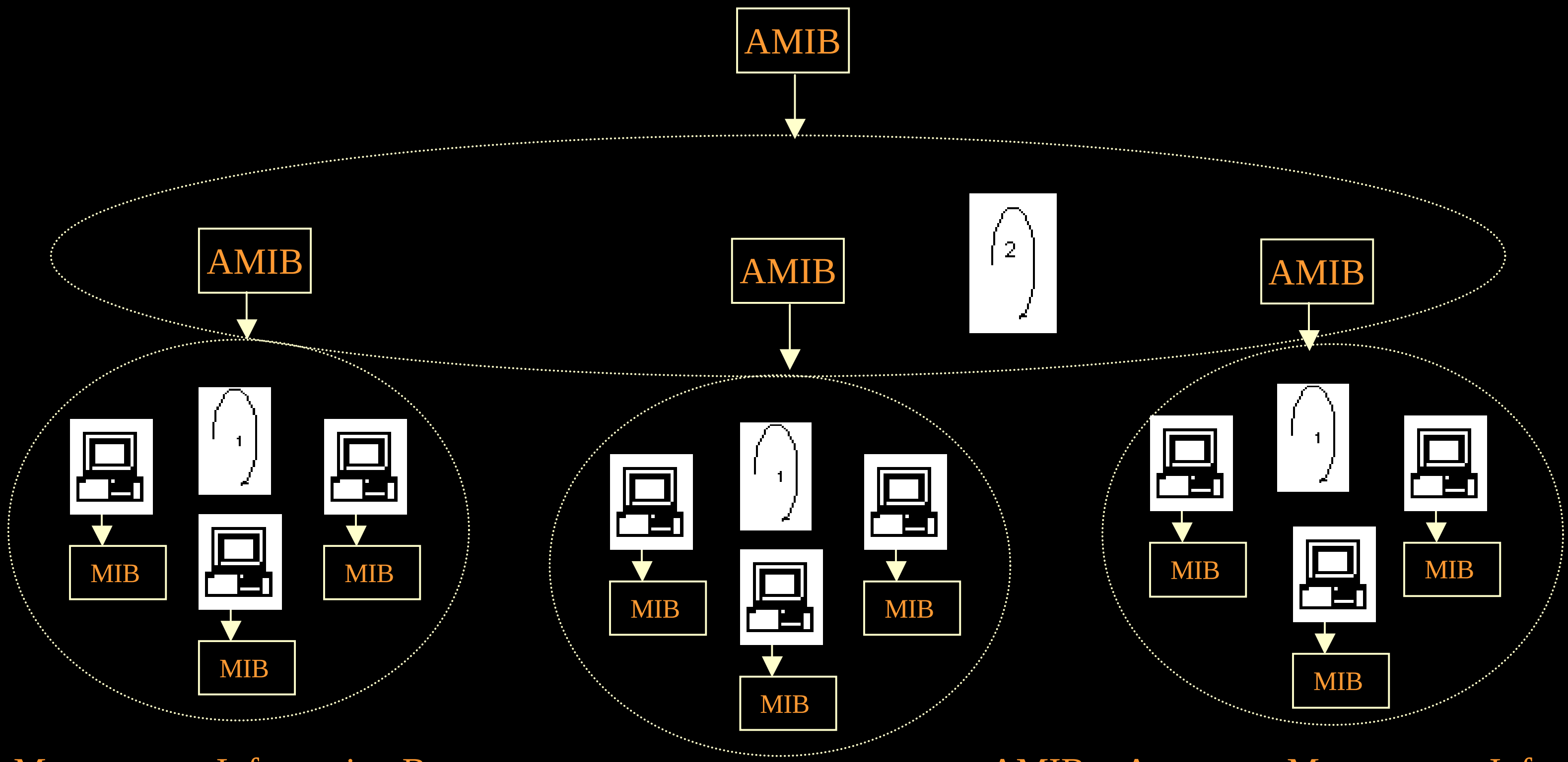
Software Architecture



- Resource monitoring service is scalable, distributed and fault-tolerant
- Simple API provided for interfacing with applications and other services such as load balancers, schedulers, etc.

Resource Monitoring Service

Structure



MIB – Management Information Base

AMIB – Aggregate Management Information Base

- Monitored parameters collectively form a management information base
- System parameters are exchanged within each group in Layer-1 while aggregate values are exchanged between groups

Resource Monitoring Service

Sample Built-in Aggregation Functions

Load Average	2.0
Load Average	1.5
Free Memory	120M
Resource Availability	Y

Mean aggregation fn

Minimum aggregation fn

Maximum aggregation fn

Boolean aggregation fn

Load Average	1.5
Free Memory	120M
Resource Availability	Y

Load Average	2.5
Free Memory	20M
Resource Availability	N

- Consistency of data maintained with heartbeat values used for failure detection
- Aggregate functions and user data can be dynamically added
- Functions and data are uniquely identified by IDs assigned by the service

Resource Monitoring Service

Sample API Functions

- API functions broadly classified into
  - Initialization functions – I
  - Control functions – C
  - Update functions – U

API Function name	Operation	Return arguments	Type
gms_init	Register RMA with gossip agent	Success/Failure	I
gms_aggfn_init	Assign ID for new aggregation function	ID assigned	I
gms_kill	Stop dissemination of monitor data	Success/Failure	C
gms_userdata_kill	Stop dissemination of user data identified by ID	Success/Failure	C
gms_recv_userdata	Receive user data from RMA	User data of nodes and aggregate data of group	U

Conclusions

- Enhancements made with efficient, scalable, and resilient low-level service for failure detection and consensus
- Targeted for heterogeneous, distributed, large-scale systems
- Tradeoffs identified in # of gossip layers versus system size
  - Scalability of consensus time and resource utilization into 1000s of nodes!
- Model to characterize impact of failure service characteristics on application performance; support comparisons
- New resource monitoring and management service with inherent and user-definable system state information
- Disseminated resource status across system with same advantages in performance, scalability, and resilience as in failure detection

Future Directions

- Investigate how best to couple failure detection service with application middleware (e.g. MPI, PVM) for cluster computing
- Investigate how best to couple resource monitoring service with prominent load balancing/scheduling services
- Investigate issues when moving from clusters to grids
- Improve dynamic system reconfiguration to become more of a plug-and-play system
- Develop GUI to dynamically render both failure and resource state for sysadmin and user usage
- Support Sandia requirements for s/w quality assurance
- Investigate use of resource monitoring service for forecasting by maintaining experiential database of values and timestamps

References

1. R. Van Renesse, R. Minsky, and M. Hayden, "A Gossip-style Failure Detection Service," *Proc. of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing Middleware*, England, September 15-18, 1998, pp. 55-70.
2. R. Van Renesse, "Scalable and Secure Resource Location," *Proc. of the IEEE Hawaii International Conference on System Sciences*, Maui, Hawaii, January 4-7, 2000.
3. I. Ahmed, "Semi-distributed Load Balancing for Massively Parallel Multicomputer Systems," *IEEE Transactions on Software Engineering*, Vol. 17, No. 10, October 1991, pp. 987-1004.
4. M. Burns, A. George, and B. Wallace, "Simulative Performance Analysis of Gossip Failure Detection for Scalable Distributed Systems", *Cluster Computing*, Vol. 2, No. 3, 1999, pp.207-217.
5. S. Ranganathan, A. George, R. Todd, and M. Chidester, "Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters," *Cluster Computing*, Vol. 4, No. 3, July 2001, pp.197-209.
6. K. Sistla, A. George, R. Todd and R. Tilak, "Performance Analysis of Flat and Layered Gossip Services for Failure Detection and Consensus in Scalable Heterogeneous Clusters," *Proc. of IEEE Heterogeneous Computing Workshop at IPDPS*, San Francisco, CA, April 23-27, 2001.
7. D. Collins, A. George, and R. Quander, "Achieving Scalable Cluster System Analysis and Management with a Gossip-based Network Service," *Proc. of IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, November 14-16, 2001.